Definition

Refactoring (Verb): Umstrukturierung von Software durch Anwendung einer Reihe von Refactorings, ohne das beobachtbare Verhalten der Software zu ändern.

Refactoring (Substantiv): eine Änderung der internen Struktur einer Software, um sie verständlicher und einfacher modifizierbar zu machen, ohne das beobachtbare Verhalten zu verändern.

— Martin Fowler

"Refactoring" hat ernsthaften Schaden genommen, als die Leute begannen, damit lange Pausen bei der Feature-Entwicklung zu erklären.

Dabei verschwand sogar der Teil mit dem »Verhalten beibehalten«, sodass ein Refactoring das System problemlos aus dem Tritt bringen kann. Keine neuen Features, möglicher Schaden und nichts, was man am Ende vorzeigen kann. Danke, ich verzichte.

— Kent Beck


REFACTORING

Aufräumereien sind eine Untermenge der Refactorings. Sie sind niedliche, knuddelige kleine Refactorings, die niemand wirklich hassen kann.

— Kent Beck


Tidyings!
Part I. Tidyings
Chapter 1. Guard Clauses
Chapter 2. Dead Code
Chapter 3. Normalize Symmetries
Chapter 4. New Interface, Old Implementation
Chapter 5. Reading Order
Chapter 6. Cohesion Order
Chapter 7. Move Declaration and Initialization Together
Chapter 8. Explaining Variables
Chapter 9. Explaining Constants
Chapter 10. Explicit Parameters
Chapter 11. Chunk Statements
Chapter 12. Extract Helper
Chapter 13. One Pile
Chapter 14. Explaining Comments
Chapter 15. Delete Redundant Comments

Guard Clauses

```
function calculateDiscount(price: number, isHoliday: boolean, isVIP: boolean) {
    let discount = 0;
    if (isHoliday) {
        if (!isVIP) {
            discount = price * 0.1;
        }
    }
    return discount;
}
```

Guard Clauses

```
function calculateDiscount(price: number, isHoliday: boolean, isVIP: boolean) {
    let discount = 0;
    if (isHoliday) {
        if (!isVIP) {
            discount = price * 0.1;
        }
    }
    return discount;
}
```

```
function calculateDiscount(price: number, isHoliday: boolean, isVIP: boolean) {
    if (!isHoliday) return 0;
    if (isVIP) return 0;

    return price * 0.1;
}
```

Dead Code

```
class BankAccount {
    private balance: number = 0;
    private transactionHistory: number[] = [];

    private calculateAverageTransaction(): number {
        const sum = this.transactionHistory.reduce(
          (a, b) => a + b, 0);
        return sum / this.transactionHistory.length;
    }

    public getBalance(): number {
        return this.balance;
    }

    public printStatementFormA(): void {
        console.log("Balance: ", this.getBalance());
        console.log("Average Transaction: ",
          this.calculateAverageTransaction());
    }

    public printStatementFormB(): void {
        console.log("Balance: ", this.getBalance());
        console.log(account.getTransactionHistory());
    }

    public deposit(amount: number): void {
        this.balance += amount;
        this.transactionHistory.push(amount);
    }

    public getTransactionHistory(): number[] {
        return this.transactionHistory;
    }

    public withdraw(amount: number): void {
        if (amount <= this.balance) {
            this.balance -= amount;
            this.transactionHistory.push(-amount);
        } else {
            console.log("Insufficient balance");
        }
    }
}

const specialTransactionPrint =
    (inValueFromT1, inValueFromT2, outValueFromT3) => {
        const account = new BankAccount();
        account.deposit(inValueFromT1);
        account.deposit(inValueFromT2);
        account.withdraw(outValueFromT3);
        printStatementFormB();
}
```

Normalize Symmetries

```java
public class Person {
...
    // Using string concatenation
    public String getFullName() {
        return firstName + " " + lastName;
    }

    // Using String.format
    public String getLocation() {
        return String.format("%s, %s", city, country);
    }

    // Using StringBuilder
    public String getDetailedInfo() {
        return new StringBuilder(firstName).append(" ").append(lastName)
                .append(" from ").append(city).append(", ").append(country).toString();
    }
}
```

```java
// Using string concatenation
public String getFullName() {
    return firstName + " " + lastName;
}
// Using String.format
public String getLocation() {
    return String.format("%s, %s", city, country);
}
// Using StringBuilder
public String getDetailedInfo() {
    return new StringBuilder(firstName).append(" ").append(lastName)
            .append(" from ").append(city).append(", ").append(country).toString();
}
```

```java
public String getFullName() {
    return firstName + " " + lastName;
}

public String getLocation() {
    return city + ", " + country;
}

public String getDetailedInfo() {
    return firstName + " " + lastName + " from " + city + ", " + country;
}
```

New Interface, Old Implementation

```
class Database:
    def connect(self, url, username, password):
        ...

    def execute_query(self, query):
        ...

    def disconnect(self):
        ...
```

```
    def connect(self, url, username, password):
    def execute_query(self, query):
    def disconnect(self):

    def execute_query(self, url, username, password, query):
        self.connect(url, username, password)
        self.execute_query(query)
        self.disconnect()
```

Usage

```
db = Database()
db.execute_query("url", "username", "password", "SELECT * FROM table")
```

Reading Order

```
class Car:
    def start_engine(self):
        if self.check_battery():
            print("Engine started.")
        else:
            print("Engine failed to start. Battery is dead.")

    def __init__(self, make, model, year, color):
        self.make = make
        self.model = model
        self.year = year
        self.color = color

    def check_battery(self):
        # For simplicity, let's assume the battery is good if the car is less than 5 years old
        return datetime.datetime.now().year - self.year < 5
```

```
    def start_engine(self):
        if self.check_battery():
            print("Engine started.")
        else:
            print("Engine failed to start. Battery is dead.")

    def __init__(self, make, model, year, color):
        ...

    def check_battery(self):
        return datetime.datetime.now().year - self.year < 5
```

```
    def __init__(self, make, model, year, color):
        ...

    def check_battery(self):
        # For simplicity, let's assume the battery is good if the car is less than 5 years old
        return datetime.datetime.now().year - self.year < 5

    def start_engine(self):
        if self.check_battery():
            print("Engine started.")
        else:
            print("Engine failed to start. Battery is dead.")
```

Cohesion Order

```
class ShoppingCart:
    def __init__(self):
        self.items = []

    def apply_discount(self, code):
        if self.is_valid_discount_code(code):
            total = self.calculate_total()
            print(f"Total after discount: {total * 0.9}")
        else:
            print("Invalid discount code.")

    def calculate_item_total(self, item):
        return item['price'] * item['quantity']

    def add_item(self, name, price, quantity):
        self.items.append({'name': name, 'price': price, 'quantity': quantity})

    def calculate_total(self):
        ...

    def is_valid_discount_code(self, code):
        return code.startswith("DISC")
```

Cohesion Order

```
    def __init__(self): ...

    def apply_discount(self, code): ...

    def calculate_item_total(self, item): ...

    def add_item(self, name, price, quantity): ...

    def calculate_total(self): ...

    def is_valid_discount_code(self, code): ...
```

```
    def __init__(self): ...

    def add_item(self, name, price, quantity): ...

    def calculate_item_total(self, item): ...

    def calculate_total(self): ...

    def is_valid_discount_code(self, code): ..

    def apply_discount(self, code): ...
```

Move Declaration an Initialization Together

```
let total: number;
// ...some code that doesn't use total
// ...some code that doesn't use total
// ...some code that doesn't use total
total = 0;
for (let item of items) {
    let price: number;
    // ...some more code, maybe it uses total but doesn't use price
    // ...some more code, maybe it uses total but doesn't use price
    // ...some more code, maybe it uses total but doesn't use price
    price = item.price;
    total += price * item.quantity;
}
return total;
```

Move Declaration an Initialization Together

```
let total: number;
// ...some code that doesn't use total
total = 0;
for (let item of items) {
    let price: number;
    // ...some more code, maybe it uses total but doesn't use price
    price = item.price;
    total += price * item.quantity;
}
return total;
```

```
let total = 0;
// ...some code that doesn't use total
for (let item of items) {
    let price: number;
    // ...some more code, maybe it uses total but doesn't use price
    price = item.price;
    total += price * item.quantity;
}
return total;
```

Explaining Variables

```
def calculate_point():
    return (math.sqrt(16) * math.sin(math.radians(45)), math.sqrt(16) * math.cos(math.radians(45)))
```

Explaining Variables

```
def calculate_point():
    return (math.sqrt(16) * math.sin(math.radians(45)), math.sqrt(16) * math.cos(math.radians(45)))
```

```
def calculate_point():
    radius = math.sqrt(16)
    angle_in_radians = math.radians(45)

    x = radius * math.sin(angle_in_radians)
    y = radius * math.cos(angle_in_radians)

    return (x, y)
```

Explaining Constants

```
function handleResponse(response: {code: number, data: any}): void {
    if (response.code === 404) {
        console.log("Page not found.");
    } else if (response.code === 200) {
        console.log("Success!");
    }
}
```

Explaining Constants

```
function handleResponse(response: {code: number, data: any}): void {
    if (response.code === 404) {
        console.log("Page not found.");
    } else if (response.code === 200) {
        console.log("Success!");
    }
}
```

```
const PAGE_NOT_FOUND = 404;
const SUCCESS = 200;

function handleResponse(response: {code: number, data: any}): void {
    if (response.code === PAGE_NOT_FOUND) {
        console.log("Page not found.");
    } else if (response.code === SUCCESS) {
        console.log("Success!");
    }
}
```

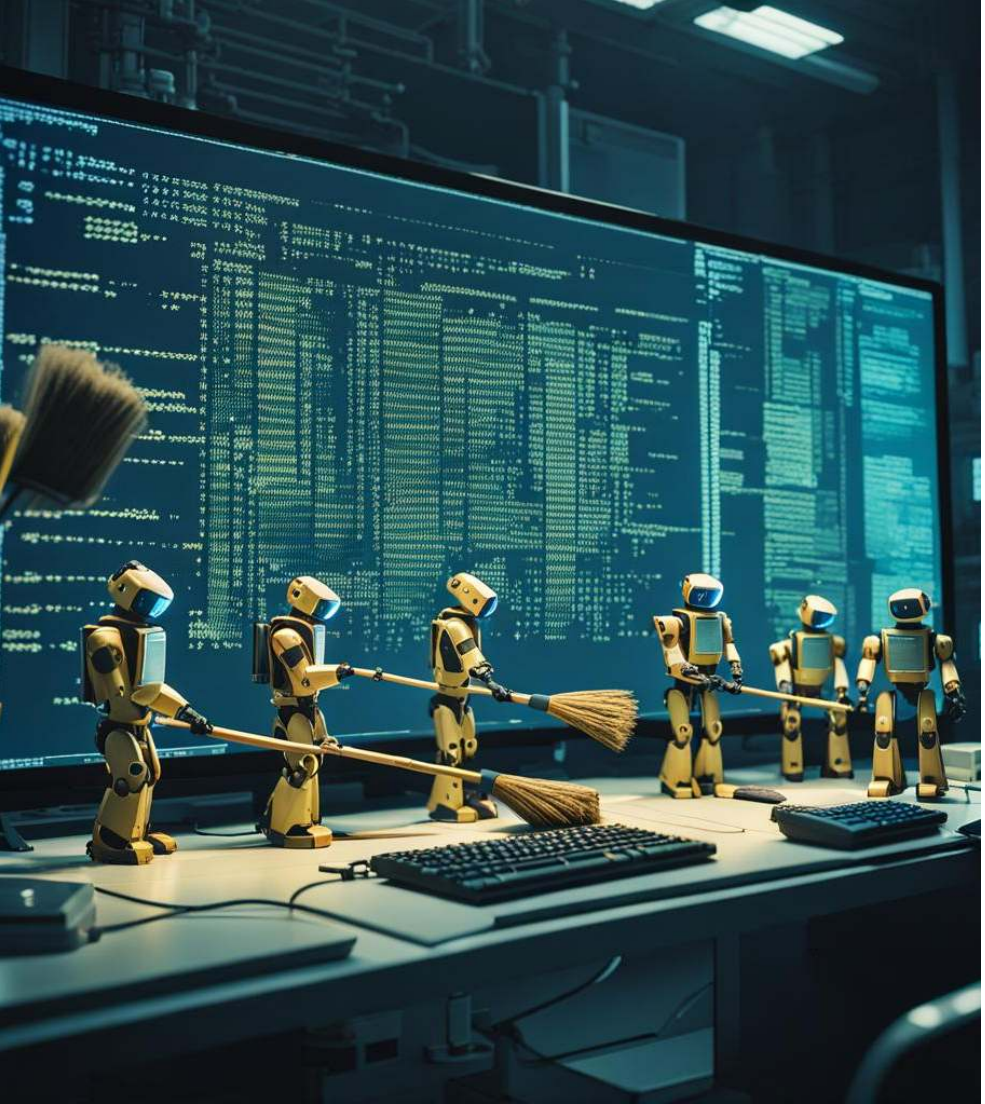
Explicit Parameters

```
public static void main(String[] args) {
    HashMap<String, Integer> params = new HashMap<>();
    params.put("width", 10);
    params.put("height", 20);
    calculateArea(params);
}
static void calculateArea(HashMap<String, Integer> params) {
    int area = params.get("width") * params.get("height");
    System.out.println("Area: " + area);
 }
```

Explicit Parameters

```java
public static void main(String[] args) {
    HashMap<String, Integer> params = new HashMap<>();
    params.put("width", 10);
    params.put("height", 20);
    calculateArea(params);
}
static void calculateArea(HashMap<String, Integer> params) {
    int area = params.get("width") * params.get("height");
    System.out.println("Area: " + area);
}
```

```java
public static void main(String[] args) {
    HashMap<String, Integer> params = new HashMap<>();
    params.put("width", 10);
    params.put("height", 20);
    calculateArea(params.get("width"), params.get("height"));
}
static void calculateArea(int width, int height) {
    int area = width * height;
    System.out.println("Area: " + area);
}
```



More Tidyings

- Chunk Statements
- Extract Helpers
- One Pile
- Explaining Comments
- Delete Redundant Comments
- even more…


Tidyings!
Part I. Tidyings
Chapter 1. Guard Clauses
Chapter 2. Dead Code
Chapter 3. Normalize Symmetries
Chapter 4. New Interface, Old Implementation
Chapter 5. Reading Order
Chapter 6. Cohesion Order
Chapter 7. Move Declaration and Initialization Together
Chapter 8. Explaining Variables
Chapter 9. Explaining Constants
Chapter 10. Explicit Parameters
Chapter 11. Chunk Statements
Chapter 12. Extract Helper
Chapter 13. One Pile
Chapter 14. Explaining Comments
Chapter 15. Delete Redundant Comments